A Simplified Quadratic Frobenius Primality Test

by Martin Seysen

December 20, 2005

Giesecke & Devrient GmbH
Prinzregentenstr. 159, D-81677 Munich, Germany
martin.seysen@gi-de.com

Abstract

The publication of the quadratic Frobenius primality test [6] has stimulated a lot of research, see e.g. [4, 10, 11]. In this test as well as in the Miller-Rabin test [13], a composite number may be declared as probably prime. Repeating several tests decreases that error probability. While most of the above research papers focus on minimising the error probability as a function of the number of tests (or, more generally, of the computational effort) asymptotically, we present a simplified variant SQFT of the quadratic Frobenius test. This test is so simple that it can easily be implemented on a smart card.

During prime number generation, a large number of composite numbers must be tested before a (probable) prime is found. Therefore we need a fast test, such as the Miller-Rabin test with a small basis, to rule out most prime candidates quickly before a promising candidate will be tested with a more sophisticated variant of the QFT. Our test SQFT makes optimum use of the information gathered by a previous Miller-Rabin test. It has run time equivalent to two Miller-Rabin tests; and it achieves a worst-case error probability of 2^{-12t} with t tests.

Most cryptographic standards require an average-case error probability of at most 2^{-80} or 2^{-100}, see e.g. [7], when prime numbers are generated in public key systems. Our test SQFT achieves an average-case error probability of 2^{-134} with two test rounds for $500-$bit primes.

We also present a more sophisticated version SQFT3 of our test that has run time and worst-case error probability comparable to the test EQFTwc presented in [4] in all cases. The test SQFT3 avoids the computation of cubic residuosity symbols, as required in the test EQFTwc.

Key Words:
smart card, prime number generation, primality testing, quadratic Frobenius test

1 Introduction

Efficient prime number tests are important in theory and practice, e.g. for RSA key generation and for testing if a system parameter in a public-key system supposed to be prime is really prime. In this paper we present a fast and simple probabilistic primality test that can be considered as a variant of the quadratic Frobenius test [6]. All such tests considered in this paper may declare a composite number prime with a small *error probability*; but they never declare a prime number composite.

There are two reasons why primality tests should be fast and simple. First it should be noted that the key size for public-key cryptosystems such as RSA is likely to increase considerably in the next ten to 20 years, see e.g. [8]. In many cases, there is also a requirement to keep the generated primes secret, so that prime number generation must run in a secure environment, such as a smart card or a shielded cryptographic device. Compared to a standard PC, the computational power of such a device is usually quite limited, so that primality testing should not take too much time or storage.

When generating prime numbers for public-key cryptosystems, a cryptographic device chooses random integers of appropriate size and tests them for primality, until a probable prime is found. Here the *average case* error probability of the test must be small. In other scenarios, public key parameters (e.g. of the Diffie-Hellman key exchange protocol) must be verified. Since these parameters may have been chosen by an adversary, the *worst-case* error probability of the primality test must be small here.

Many applications use the Miller-Rabin test as the standard tool for primality testing. The worst-case error probability of the Miller-Rabin test is $\leq 4^{-t}$ when performing t tests, see [9]. Using different variants of the QFT, we asymptotically obtain a better error probability for the same computational effort.

In case of prime n, the ring $\mathbb{Z}_n$ of integers modulo n is equivalent to the Galois field $GF(n)$, its multiplicative group $\mathbb{Z}_n^*$ is cyclic of order $n-1$, and there is exactly one primitive 2^{nd} root of unity, namely -1. If any of these properties is violated, n cannot be prime. The basic idea of the Miller-Rabin test is to pick a random element of $\mathbb{Z}_n^*$ and to check if it has order dividing $n-1$. Furthermore, some calculations in the 2-Sylow group of $\mathbb{Z}_n^*$ are performed in order to check if square roots of 1 apart for ± 1 are present, as this is the case for most composite n.

Most variants of the QFT work with the ring $\mathbb{Z}_n[x]/f(x)$, where $f(x)$ is a quadratic polynomial over $\mathbb{Z}_n$. This ring is equivalent to the Galois field $GF(n^2)$, if n is prime and $f(x)$ is irreducible over $\mathbb{Z}_n$. Fortunately, such polynomials are easy to find: A quadratic polynomial over $\mathbb{Z}_n$, n prime, is irreducible if and only if its discriminant is a quadratic nonresidue modulo n. Again, a variety of properties must hold in $GF(n^2)$, n prime, and can be checked for random elements in $(\mathbb{Z}_n[x]/f(x))^*$. First of all, $GF(n^2)$ is cyclic of order n^2-1, so any $z \in (\mathbb{Z}_n[x]/f(x))^*$ must have an order dividing n^2-1. Also, there is a natural automorphism, called *conjugation* in $\mathbb{Z}_n[x]/f(x)$, that must be equivalent to the Frobenius automorphism $z \rightarrow z^n$ in $GF(n^2)$ for prime n. Furthermore, if n is not divisible by 2 or 3, then n^2-1 is divisible by 24, and in case of prime n, the field $GF(n^2)$ has a cyclic group of 24^{th} roots of unity, generated by a primitive root. All primitive $q-$th roots of unity z, if they exist, must satisfy $\Phi_q(z) = 0$, for the $q-$th cyclotomic

polynomial Φ_q. For technical reasons, 4^{th} (or 8^{th}) and 3^{rd} roots of unity are checked separately. Therefore, calculations in the 2- and 3-Sylow group of $(\mathbb{Z}_n[x]/f(x))^*$ are necessary.

Some or all of the above properties are checked by the different variants of the QFT. The quadratic polynomial $f()$ may either be chosen at random or fixed. In case of a random polynomial, one may average over certain suitable polynomials to decrease the error probability. In case of a fixed polynomial, one may collect roots of unity, when they appear during the calculations, and check if the group of all roots found so far is still cyclic. Both approaches have advantages and disadvantages. It turns out that for most variants of the QFT, the cost for a test round is essentially the cost of an exponentiation in $\mathbb{Z}_n[x]/f(x)$ with an exponent of size n. The cost for a test round of the Miller-Rabin test is essentially the cost of an exponentiation in $\mathbb{Z}_n$ with an exponent of the same size. One of the more important advantages of taking a fixed polynomial $f()$ with small coefficients is that a test round of the QFT has about twice the run time of a test round of the Miller-Rabin test. For the QFT with a random polynomial, that ratio of run times is about three.

In the original Quadratic Frobenius Test proposed by Grantham [6], the quadratic polynomial $f()$ has been chosen at random. The worst-case error probability is about 7710^{-t} for t test rounds, and the run time is equivalent to about three Miller-Rabin tests per test round. Müller [10, 11] has proposed a test using 3^{rd} roots of unity with worst-case error probability about 131000^{-t} and, asymptotically, the same run time as the original QFT. Depending on n mod 24, better bounds are achieved in some cases.

Damgård and Frandsen [4] have proposed a test EQFTwc (optimised for the worst case) with a fixed polynomial and worst-case error probability $256 \cdot 576^{-2t}$ and cost equivalent to 2 Miller-Rabin tests per test round. However, this test requires a precomputation of cubic residuosity symbols, at a cost of about 2 Miller-Rabin tests, to ensure that the 3-Sylow group of $(\mathbb{Z}_n[x]/f(x))^*$ has a suitable structure. As we shall see later, each cyclic factor of that 3-Sylow group is associated with a prime p dividing n. The purpose of the precomputations is to ensure that all these factors are large enough in the sense that they are at least as large as the 3-Sylow group of the cyclic group of order $n^2 - 1$.

In [4], another test EQFTac (optimised for the average case) has been proposed, and its average-case complexity has been analysed. This test does not require a precomputation, but it also does not achieve a run time equivalent to two Miller-Rabin tests in all cases. In this test, nothing must be known about the 3-Sylow group a priori. On the other hand, if $n^2 - 1$ is divisible by a high power 2 and of 3, the necessary calculations are asymptotically more expensive than for Algorithm EQFTwc.

So, when dealing with 3^{rd} roots of unity, we may either check by a precomputation, that all factors of the 3-Sylow group of $(\mathbb{Z}_n[x]/f(x))^*$ are sufficiently large, or we have to do some extra calculations in that 3-Sylow group during each test round, in order to find 3^{rd} roots of unity.

A similar alternative exists for the 8^{th} roots of unity and the computations in the 2-Sylow group of $(\mathbb{Z}_n[x]/f(x))^*$. It turns out that all necessary computations concerning the 2-Sylow group of $(\mathbb{Z}_n[x]/f(x))^*$ can be performed in the first test round. In our

Table 1: Worst-case error probability for some primality tests at an effort of t Miller-Rabin tests

test	error probability on effort t	precomputation effort	effort per test round
Miller-Rabin	4^{-t}	0	1
Grantham [6]	$O(19.8^{-t})$		3
Müller [10, 11]	$O(50.8^{-t})$		3
EQFTwc [4]	$\approx 256 \cdot 576^{2-t}$	≈ 2	2
Our test SQFT	$\approx 64^{1-t}$	≈ 1	2
Our test SQFT3	$\approx 16 \cdot 576^{1-t}$	≈ 1	2

new primality test an additional precomputation equivalent to a single Miller-Rabin test with a small basis will give us either a primitive $8 - th$ root of unity in a suitable quadratic extension of $\mathbb{Z}_n$ or a proof that n is composite.

In case of prime number generation, the cost for an initial Miller-Rabin test can be neglected for a random test candidate, since such a random candidate is likely to be composite, and we need a fast test to rule out composite candidates anyway. Even if many composite candidates have been ruled out in a prime number generation algorithm by sieving or trial division, the probability that one of the remaining candidates is prime is yet considerably smaller than 1/2 in cases of cryptographical interest.

In the following two sections we present a very simple Algorithm SQFT based on this idea. In order to keep the exposition simple, we ignore the 3^{rd} roots of unity in that algorithm.

In section 4 we add a 3^{rd}-root-of-unity test to Algorithm SQFT, leading to Algorithm SQFT3. In section 5 we will bound the run time of Algorithm SQFT3. It turns out that the run time of Algorithm SQFT3 is asymptotically the same as for the Algorithms SQFT and Algorithm EQFTac in all cases, which is equivalent to about two Miller-Rabin tests per test round.

In the following three sections we estimate the worst-case error probability and the average-case error probability for both, random search and incremental search. Here we use the same basic idea as in [4]. Since the focus of our paper is to find a simple variant of the QFT, we analyse the average-case error probability of Algorithm SQFT. It turns out that in cases of cryptographic interest, the requirements in [7] are met with two test rounds of Algorithm SQFT. We also believe that the additional effort of Algorithm SQFT3, although it can be neglected asymptotically, is yet considerable in cases of cryptographical interest.

Table 1 shows the worst-case error probabilities for some variants of the QFT. In Table 1, effort t corresponds to the effort required for t Miller-Rabin tests; the last two results in the table are proved in section 6. We also list the precomputation effort and the effort required per test round (in units counting the effort for one Miller-Rabin test) for some of the tests.

2 Outline of the Simplified Quadratic Frobenius Test

Let $\mathbb{Z}_n$ denote the ring of integers modulo n. For $k, n \in \mathbb{Z}, n$ odd, let (k/n) be the Jacobi symbol. We write (a, b) for $\gcd(a, b)$. For any set S we write $|S|$ for its cardinality. Throughout the paper $\mathbb{P}(n)$ denotes the set of primes dividing n for any integer n. For any integer n and prime p let $v_p(n)$ be the exponent m of the highest power p^m of p dividing n. Thus $|\mathbb{P}(n)|$ is the number of different prime factors of n and $\prod_{p\in\mathbb{P}(n)} p^{v_p(n)}$ is the *prime power factorisation* of n. We write Φ_i for the i-th cyclotomic polynomial. We mainly deal with $\Phi_3(z) = z^2 + z + 1$ and $\Phi_8(z) = z^4 + 1$. The number n will always be the (odd) integer to be tested for primality.

Definition 1 *For an odd natural number n and a unit c modulo n, let $R(n, c)$ denote the ring $\mathbb{Z}[x]/(n, x^2 - c)$.*

We represent each element of $R(n, c)$ as a polynomial $ax + b$ of degree 1 in the variable x with $0 \le a, b < n$. We write $R(n, c)^*$ for the multiplicative group of units in $R(n, c)$.

Definition 2 *For $z = ax + b$ we define the following multiplicative homomorphisms in $R(n, c)$:*

$$\begin{array}{lll} \bar{\cdot} : & R(n,c) \to R(n,c)\ , & \bar{z} = b - ax \quad ; \\ N(\cdot): & R(n,c) \to \mathbb{Z}_n\ , & N(z) = \bar{z}\cdot z = b^2 - ca^2 \quad . \end{array}$$

As usual, these two homomorphisms are called *conjugation* and *norm*. Note that conjugation actually is an automorphism in $R(n, c)$.

The following algorithm performs the necessary preparations before starting the simplified quadratic Frobenius test for a candidate n. Its run time is essentially the run time for a Miller-Rabin test with a small basis. The algorithm either outputs a small quadratic nonresidue c modulo n and a primitive 8^{th} root of unity in $R(n, c)$ or proves that n is composite.

Algorithm MR2 *(Miller-Rabin test with basis two or a small nonresidue)*

`Input` An odd integer n .
`Output` Either the statement "n is composite" or an integer c
with $(c/n) = -1$ and $\epsilon \in R(n, c)$ with $\epsilon^4 = -1$ (i.e. $\Phi_8(\epsilon) = 0$) .

[1] `If` $n = 3 \pmod 4$ `then` do the following:
Compute $\alpha = 2^{\frac{n-3}{4}} \pmod n$.
`If` $2\alpha^2 \neq \pm 1 \pmod n$ `then output` "n is composite" and `stop` ;
`else output` $c = -1, \epsilon = \alpha + \alpha x$ and `stop` .
[2] `If` $n = 5 \pmod 8$ `then` do the following:

Compute $\alpha = 2^{\frac{n-1}{4}} \pmod n$.
`If` $\alpha^2 \neq -1 \pmod n$ `then output` "n is composite" and `stop` ;
`else output` $c = 2, \epsilon = \frac{1+\alpha}{2}x$ and `stop` .

[3] `If` $n = 1 \pmod 8$ `then` do the following:
`If` n is a perfect square `then output` "n is composite" and `stop` .
Compute a small random value c with $(c/n) = -1$.
// (We will give a more precise definition of "small" in section 5.)
Compute $\alpha = c^{\frac{n-1}{8}} \pmod n$.
`If` $\alpha^4 \neq -1 \pmod n$ `then output` "n is composite" and `stop` ;
`else output` $c, \epsilon = \alpha$ and `stop` .

Note that Algorithm MR2 actually performs a Miller-Rabin test with basis 2 in case $n \neq 1 \pmod 8$ and with basis c in case $n = 1 \pmod 8$. The property $\epsilon^4 = -1 \pmod{R(n,c)}$ is obvious in case $n = 1 \pmod 8$ and easy to verify by using the standard representations $(\pm 1 \pm \sqrt{-1})/\sqrt{2}$ of the four primitive 8-th roots of unity in the other cases.

The following algorithm performs a single round of the simplified quadratic Frobenius test.

Algorithm SQFT$_{\text{round}}$

`Input` An odd integer n , a small integer c with $(c/n) = -1$,
and a value $\epsilon \in R(n,c)$ with $\epsilon^4 = -1$, (i.e. $\Phi_8(\epsilon) = 0$).
`Output` Either the statement "n is composite" or
the statement "n is possibly prime".

[1] Select a random $z \in R(n,c)$ with $(N(z)/n) = -1$.
[2] `If` $z^n \neq \bar{z}$ `then output` "n is composite" and `stop` .
[3] `If` $z^{\frac{n^2-1}{8}} \notin \{\pm\epsilon, \pm\epsilon^3\}$ `then output` "n is composite" and `stop` .
[4] `output` "n is possibly prime" and `stop` .

Now we present the simplified quadratic Frobenius test. Here we first apply Algorithm MR2. This gives us an 8-th root of unity with the effort of essentially one Miller-Rabin test. Then we repeatedly apply Algorithm SQFT$_{\text{round}}$ in order to reduce the probability that a composite n is declared prime.

Algorithm SQFT *(Simplified Quadratic Frobenius Test)*

`Input` An integer n with $n > 20$ and a number t of test rounds.
`Output` Either the statement "n is composite"
or the statement "n is probably prime".

[1] `If` n is divisible by a prime $p < 20$ `then output` "n is composite" and `stop` .

[2] `Call` Algorithm MR2 with input n.
[3] `If` Algorithm MR2 declares n composite `then stop` .
[4] Let c, ϵ be the output of Algorithm MR2 if n has been declared prime.
[5] `For` $i = 1, \ldots, t$ do the following:
`Call` Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ with input n, c, ϵ.
`If` Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ declares n composite `then stop` .
[6] `output` "n is probably prime" and `stop` .

Remarks

It easy to check that neither Algorithm MR2 nor Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ will ever declare a prime n composite.

After the first test round we may further simplify Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ without deteriorating the error probability, see section 5 for details.

One might argue that the introductory Miller-Rabin test costs some extra effort, but gives us little benefit compared to the extended quadratic Frobenius test in [4]. However, when generating primes, we have to check many candidates, and only a small fraction of candidates will eventually turn out to be prime. So we need a fast test that quickly rules out most composite candidates. Note that the Miller-Rabin test is at least twice as fast as any known variant of the QFT and rules out almost all composite candidates in the cases of practical interest. So, anyway, it makes sense to subject all candidates to a Miller-Rabin test before using more sophisticated primality tests for prime number generation. In Algorithm SQFT we exploit the information gained during this introductory Miller-Rabin test phase for simplifying the subsequent quadratic Frobenius test.

3 A Bound for the Fraction of Liars

In this section we give an upper bound for the probability that Algorithm SQFT declares a composite number prime.

Definition 3 *For any composite number n we define $\beta_{1,t}(n)$ as the probability that Algorithm SQFT (when running with t test rounds) declares the number n prime.*

For composite n we define $\beta_{1,t}(n, c)$ as the probability that Algorithm SQFT (when running with t test rounds) declares n prime, under the condition that Algorithm MR2 declares n prime and outputs c.

If Algorithm MR2 never outputs c on input n, we define $\beta_{1,t}(n, c) = 0$.

Obviously, we have:

$$\beta_{1,t}(n) \leq \max_{(c/n)=-1} \beta_{1,t}(n, c) \ .$$

The purpose of this section is to show:

Proposition 4 *Let $\beta_{1,t}(n,c)$ be defined as above. Then $\beta_{1,t}(n,c)$ is at most $(\beta_1(n,c))^t$, where $(\beta_1(n,c))$ is given by:*

$$8^{1-|\mathbb{P}(n)|} \cdot \prod_{\substack{p\in\mathbb{P}(n),\\ (c/p)=-1}} p^{2-2v_p(n)}\frac{(n/p-1,p^2-1)}{p^2-1} \cdot \prod_{\substack{p\in\mathbb{P}(n),\\ (c/p)=1}} p^{2-2v_p(n)}\frac{(n^2/p^2-1,p-1)}{(p-1)^2} \quad .$$

The proof of this proposition is similar to the proof of Lemma 10 in [4]. In order to keep this paper reasonably self-contained, we give the details of the proof here. We use a slight modification of the notation in [4] for the proof.

We write $\mathrm{ord}_G(x)$ for the order of an element x in a group G.

Let c be a unit in $\mathbb{Z}_n$. We define the following subsets of $R(n,c)^*$:

$$\begin{aligned} U_\nu(n,c) &= \{z\in R(n,c)^* \mid (N(z)/n)=\nu\} \quad \text{for} \quad \nu=\pm 1 \quad ; & (1)\\ G_{q,\epsilon}(n,c) &= \left\{z\in R(n,c)^* \mid \bar z = z^n \text{ and } z^{(n^2-1)/q}=\epsilon\right\}, & (2)\\ &\quad \text{for} \quad q\in\mathbb{N}, \epsilon\in R(n,c)^* \quad \text{with} \quad q|(n^2-1)\,,\ \epsilon^q=1 \quad . \end{aligned}$$

The sets $U(n,c)$ and $G(n,c)$ in [4] correspond to $U_1(n,c)$ and $G_{2,1}\cap U_1(n,c)$ in our notation. For fixed q,ϵ,ν, the sets $G_{q,\epsilon}$ and U_ν are either empty or cosets of the groups $G_{q,1}$ and U_1, respectively, in $R(n,c)^*$. Note that $G_{q,1}(n,c)\subset G_{q',1}(n,c)$ for $q'|q$.

From these definitions we immediately obtain:

$$\beta_{1,1}(n,c) \le \max_{\epsilon\in R(n,c)^*,\,\epsilon^4=-1} \frac{\left|\left(\cup_{\epsilon'\in\{\pm\epsilon,\pm\epsilon^3\}} G_{8,\epsilon'}(n,c)\right)\cap U_{-1}(n,c)\right|}{|U_{-1}(n,c)|} \quad . \qquad (3)$$

In order to analyse the structure of $G_{8,\epsilon'}(n,c)$ and $U_{-1}(n,c)$, we factor $R(n,c)$ by using Chinese remaindering as in [4].

Let $\mathbf{Z}_n$ be the cyclic group of order n (not to be confused with the ring $\mathbb{Z}_n$ of integers modulo n). In [4], Lemma 7, the structure of $R(n,c)^*$ is given as follows:

Lemma 5 *If n is odd and has the prime power factorisation $n=\prod_{i=1}^{\omega} p_i^{m_i}$, and c is a unit modulo n then*

$$R(n,c)^* \cong R(p_1^{m_1},c)^*\times\ldots\times R(p_\omega^{m_\omega},c)^* \quad ,$$

and for all prime powers p^m dividing n we have:

$$R(p^m,c)^* \cong \mathbf{Z}_{p^{m-1}}\times\mathbf{Z}_{p^{m-1}}\times R(p,c)^* \quad .$$

In case $(c/p)=-1$ we have

$$R(p,c)^* \cong \mathbf{Z}_{p^2-1} \quad \textit{and} \quad \bar z = z^p \textit{ for } z\in R(p,c) \quad .$$

In case $(c/p)=1$ we have

$$R(p,c)^* \cong \mathbf{Z}_{p-1}\times\mathbf{Z}_{p-1} \quad \textit{and} \quad z^p=z\,,\ \overline{(z_1,z_2)}=(z_2,z_1)$$
$$\textit{for } z=(z_1,z_2)\in R(p,c) \quad .$$

For any prime power p^m with $p^m|n$ there is a natural homomorphism ϕ_{n,p^m} from $R(n,c)^*$ onto $R(p^m,c)^*$ defined by taking coefficients modulo p^m. We explicitly state that ϕ_{n,p^m} operates on the multiplicative group $R(n,c)^*$ of $R(n,c)$. Note that conjugation commutes with ϕ_{n,p^m} and $N(\phi_{n,p^m}(z)) = N(z) \pmod{p^m}$ holds for all $z \in R(p^m,c)^*$. From the proof of the lemma in [4] it is also clear that ϕ_{n,p^m} maps all direct factors of $R(n,c)^*$ apart from $R(p^m,c)^*$ to the trivial group, if $m = 1$ or $m = v_p(n)$.

Lemma 6

$$|U_1(n,c)| \;=\; |U_{-1}(n,c)| \;=\; \frac{1}{2}\,|R(n,c)^*| \quad .$$

Proof

Since $U_1(n,c)$ and, possibly, $U_{-1}(n,c)$ are the only cosets of the group $U_1(n,c)$ in $R(n,c)^*$, it suffices to show that $U_{-1}(n,c)$ is not empty. This is shown in [4], Lemma 20.

□

Lemma 7 *Let ϕ be the mapping defined by:*

$$\phi = \phi_{n,p_1} \times \cdots \times \phi_{n,p_\omega} \quad : \quad R(n,c)^* \longrightarrow R(p_1,c)^* \times, \ldots, \times R(p_\omega,c)^* \quad ,$$

where $p_1, ..., p_\omega$ are the prime factors of n. Then $ker\phi \cap G_{1,1}(n,c) = \{1\}$.

Proof

Let p^m be a prime power factor of n. Since $(p, n^2-1) = 1$, the equation $z^{n^2-1} = 1$ has exactly one solution z in $\mathbf{Z}_{p^{m-1}}$. Since $R(p^m,c)^* \cong \mathbf{Z}_{p^{m-1}} \times \mathbf{Z}_{p^{m-1}} \times R(p,c)^*$ by Lemma 5, we conclude that the system $z \in R(p^m,c)^*, z = 1 \pmod{p}, z^{n^2-1} = 1$ also has exactly one solution z in $R(p^m,c)^*$. By Chinese remaindering we obtain that the system

$$z \in R(n,c)^*\,,\; z = 1 \pmod{p_i}\,,\; z^{n^2-1} = 1\;;\; i = 1, \cdots, \omega$$

has exactly one solution in z in $R(n,c)^*$. Since any $z \in G_{1,1}(n,c) \cap \ker\phi$ must solve this system of equations, we obtain $\ker\phi \cap G_{1,1}(n,c) = \{1\}$.

□

We also show:

Lemma 8

$$\beta_{1,1}(n,c) \leq 8 \cdot \prod_{p \in \mathbb{P}(n)} p^{2-2v_p(n)} \frac{|\phi_{n,p}\left(G_{8,1}(n,c)\right)|}{|R(p,c)^*|}$$

Proof

Let ϕ be the mapping in Lemma 7. Replacing the cosets $G_{8,\epsilon}(n,c)$ by $G_{8,1}(n,c)$ in (3) we obtain:

$$\begin{aligned}
\beta_{1,1}(n,c) &\leq 8 \cdot \frac{|G_{8,1}(n,c)|}{|R(n,c)^*|} \quad ; \text{ by Lemma 6 and (3)}, \\
&\leq 8 \cdot \prod_{p \in \mathbb{P}(n)} \frac{|\phi_{n,p} G_{8,1}(n,c)|}{|R(p^{v_p(n)},c)^*|} \quad ; \text{ by Lemma 5, since } \ker\phi \cap G_{8,1}(n,c) = 1, \\
&= 8 \cdot \prod_{p \in \mathbb{P}(n)} p^{2-2v_p(n)} \frac{|\phi_{n,p} G_{8,1}(n,c)|}{|R(p,c)^*|} \quad ; \text{ by Lemma 5 .}
\end{aligned}$$

□

Lemma 9 *If p is a prime factor of n then $\phi_{n,p}(G_{q,1}(n,c))$ is cyclic and*

$$|\phi_{n,p}(G_{q,1}(n,c))| \text{ divides } \begin{cases} \gcd(n/p-1,(p^2-1),(n^2-1)/q) & \text{if } (c/p)=-1\,, \\ \gcd(p-1,(n^2-1)/q) & \text{if } (c/p)=1\,. \end{cases}$$

Proof

We write G for $\phi_{n,p}(G_{q,1}(n,c))$.

First consider the case $(c/p) = -1$. For any $z \in G$ we have $z^n = \bar{z} = z^p$ and $|R(p,c)^*| = p^2 - 1$ by Lemma 5. From this and $z^{(n^2-1)/q} = 1$ we obtain

$$\mathrm{ord}_G(z) \mid \gcd(n-p,(n^2-1)/q,p^2-1) = \gcd(n/p-1,(n^2-1)/q,p^2-1) \quad .$$

The lemma follows, since $G \subset R(p,c)^*$ and $R(p,c)^*$ is cyclic by Lemma 5.

Next, consider the case $(c/p) = 1$. We have $G \subset R(p,c)^* \cong \mathbf{Z}_{p-1} \times \mathbf{Z}_{p-1}$ and for any $z \in G$ represented as $z = (w_1, w_2)$ in $\mathbf{Z}_{p-1} \times \mathbf{Z}_{p-1}$ we have $\bar{z} = (w_2, w_1)$ by Lemma 5. Since $(w_2, w_1) = \bar{z} = z^n = (w_1^n, w_2^n)$ by definition of G , we see that w_2 is uniquely defined by w_1, implying that G is cyclic of order dividing $p-1$. $|G|$ also divides $(n^2-1)/q$ by definition of G.

□

Lemma 10 *If $G_{8,\epsilon}(n,c)$ is not empty for some ϵ with $\epsilon^4 = -1$ then*

$$|\phi_{n,p}(G_{8,1}(n,c))| \text{ divides } \begin{cases} \gcd(n/p-1,p^2-1)/8 & \text{if } (c/p)=-1\,, \\ \gcd(n^2/p^2-1,p-1)/8 & \text{if } (c/p)=1\,. \end{cases}$$

Proof

For any $z \in G_{8,\epsilon}(n,c) \subset G_{1,1}(n,c)$ we have $z^{(n^2-1)/2} = \epsilon^4 = -1$. Thus for $z_1 := \phi_{n,p}(z) \in \phi_{n,p}(G_{1,1}(n,c))$ we also have $z_1^{(n^2-1)/2} = -1$. Since -1 has order two in $\phi_{n,p}(G_{1,1}(n,c))$ we conclude $v_2(\mathrm{ord}_{\phi_{n,p}(G_{1,1}(n,c))}(z_1)) = v_2(n^2-1)$, and hence $v_2(n^2-1) \leq v_2(|\phi_{n,p}(G_{1,1}(n,c))|)$. By Lemma 9 we obtain $v_2(n^2-1) \leq v_2(\gcd(n/p-1,p^2-1))$ in case $(c/p) = -1$ and $v_2(n^2-1) \leq v_2(p-1)$ in case $(c/p) = 1$. Now the lemma follows from Lemma 9.

□

Proof of Proposition 4.

Obviously, $\beta_{1,t}(n,c) = (\beta_1(n,c))^t$ holds. Now Proposition 4 follows from Lemma 5, 8 and 10.

□

4 Adding a 3rd-root-of-unity Test

In order to add a 3rd-root-of-unity test we modify Algorithm SQFT as follows:

Algorithm SQFT3$_{\text{round}}$

`Input` An integer n with $\gcd(n,6) = 1$, a small integer c with $(c/n) = -1$,
a value $\epsilon \in R(n,c)$ with $\epsilon^4 = -1$,
and a value $\epsilon_3 \in R(n,c)$ with either $\epsilon_3 = 1$ or $\Phi_3(\epsilon_3) = 0$.

`Output` Either the statement "n is composite" or the statement
"n is possibly prime" together with a value ϵ'_3 satisfying $\epsilon'^3 = 1$ or
$\Phi_3(\epsilon'_3) = 0$. In case $\epsilon_3 \neq 1$, also $\epsilon'_3 = \epsilon_3^{\pm 1}$ holds on output.

[1] Select an random $z \in R(n,c)$ with with $(N(z)/n) = -1$.
[2] `If` $z^n \neq \bar{z}$ `then output` "n is composite" and `stop` .
[3] `If` $z^{\frac{n^2-1}{8}} \notin \{\pm\epsilon, \pm\epsilon^3\}$ `then output` "n is composite" and `stop` .
[4] Put $u = v_3(n^2-1)$, and r such that $n^2 - 1 = 3^u r$ holds.
[5] Put $i := \min\{j : 0 \leq j \leq u\,,\ z^{3^j r} = 1\}$.
// Remark: This minimum exists, since $z^{n^2-1} = z^{3^u r} = 1$ holds here.
[6] `If` $i = 0$ `then output` "n is possibly prime" and $\epsilon'_3 = \epsilon_3$ and `stop` .
[7] Put $\epsilon'_3 = z^{3^{i-1}r}$. // Then ϵ'_3 is a non-trivial 3rdroot of unity.
[8] `If` $\epsilon_3 = 1$ and $\Phi_3(\epsilon'_3) \neq 0$ `then output` "n is composite" and `stop` .
[9] `If` $\epsilon_3 \neq 1$ and $\epsilon'_3 \neq \epsilon_3^{\pm 1}$ `then output` "n is composite" and `stop` ;
[10] `output` "n is possibly prime" and ϵ'_3 and `stop` .

The simplified quadratic Frobenius test with a 3rd-root-of-unity test runs as follows:

Algorithm SQFT3 *(SQFT with 3rd-root-of-unity test)*

`Input` An odd integer n with $n > 200$ and a number t of test rounds.

`Output` Either the statement "n is composite"
or the statement "n is probably prime".

[1] `If` n is divisible by a prime $p < 200$ `then output` "n is composite" and `stop` .
[2] `Call` Algorithm MR2 with input n.
[3] `If` Algorithm MR2 declares n composite `then stop` .
[4] Let c, ϵ be the output of Algorithm MR2 if n is declared prime, and put $\epsilon_3 = 1$.

[5] `For` $i = 1, \ldots, t$ do the following:
`Call` Algorithm SQFT3$_{\text{round}}$ with input $n, c, \epsilon, \epsilon_3$.
`If` Algorithm SQFT3$_{\text{round}}$ declares n composite `then stop` .
Put $\epsilon_3 = \epsilon_3'$, where ϵ_3' is the output of Algorithm SQFT3$_{\text{round}}$ when n has been declared prime.
[6] `output` "n is probably prime" and `stop` .

Definition 11 *For any composite number n we define $\beta_{3,t}(n)$ as the probability that Algorithm SQFT3 (when running with t test rounds) declares the number n prime.*

For composite n we define $\beta_{3,t}(n, c)$ as the probability that Algorithm SQFT3 (when running with t test rounds) declares n prime, under the condition that Algorithm MR2 declares n prime and outputs c.

If Algorithm MR2 never outputs c on input n, we define $\beta_{3,t}(n, c) = 0$.

Obviously, we have:

$$\beta_{3,t}(n) \leq \max_{(c/n)=-1} \beta_{3,t}(n, c) \ .$$

Let S_2, S_3 be the 2- and 3-Sylow subgroups of $G_{1,1}(n, c)$, respectively, with $G_{1,1}(n, c)$ being the group defined by (2) .

Let $\mathcal{PR}_3 = \mathcal{PR}_3(\epsilon_3)$ be the probability that Algorithm SQWT3$_{\text{round}}$ declares n composite, if the auxiliary input ϵ_3 is given, and let $\mathcal{PR}_{31}$ be the probability that Algorithm SQFT3$_{\text{round}}$ outputs $\epsilon' = 1$, both under the condition that n has not been declared composite in steps 1 to 5. Then we have

Lemma 12

$$\mathcal{PR}_3(\epsilon) \leq \begin{cases} |S_3|^{-1} & \text{If } \Phi(z) = 0 \text{ has no solution } z \text{ in } G_{1,1}(n, c). \\ 3^{-|\mathbb{P}(n)|}(1 + 2^{|\mathbb{P}(n)|}) & \text{If } \Phi(z) = 0 \text{ has a solution in } G_{1,1}(n, c) \text{ and } \epsilon_3 = 1 \ , \\ & \quad \text{(and in this case also } \mathcal{PR}_{31} \leq 3^{-|\mathbb{P}(n)|} \text{ holds).} \\ 3^{1-|\mathbb{P}(n)|} & \text{If } \Phi(z) = 0 \text{ has a solution in } G_{1,1}(n, c) \text{ and } \epsilon_3 \neq 1 \ . \end{cases}$$

Furthermore, $|S_3| = \prod_{p \in \mathbb{P}(n)} 3^{a_p}$, *with* $a_p = v_3\left(|\phi_{n,p}(G_{1,1}(n, c))|\right)$.

Proof

Let z be the value chosen in $R(n, c)^*$ in Algorithm SQFT3$_{\text{round}}$ and assume that n has not been declared composite in steps 1 to 5. Then $z^n = \bar{z}$ and $z^{n^2-1} = 1$ hold and hence $z \in G_{1,1}(n, c)$ by (2). $G_{1,1}(n, c)$ has the factorisation $G_{1,1}(n, c) \cong S_2 \times S_3 \times H$ for some group H, so that we may write:

$$z = (z_2, z_3, z_0) \in S_2 \times S_3 \times H \cong G_{1,1}(n, c) \quad .$$

By Lemma 7 and 9, S_3 is a direct product

$$S_3 = C_1 \times, \ldots, \times C_\omega$$

of cyclic 3-groups, where $C_i = \phi_{n,p_i}(S_3)$, and $p_1, \cdots, p_\omega$, with $\omega = |\mathbb{P}(n)|$, are the different prime factors of n. The formula for $|S_3|$ follows from the fact that C_i is the 3-Sylow group of the cyclic group $\phi_{n,p_i}(G_{1,1}(n,c))$.

Furthermore, z satisfies $z^{(n^2-1)/8} = \epsilon$ and $(z/n) = -1$. But since these conditions depend only on the component z_2 of z, we may assume that z_3 is equidistributed in S_3. Note that $y := z^r$ satisfies $y^{3^u} = z^{n^2-1} = 1$ and hence $y \in S_3$. So y depends only on z_3. Since $3 \not| r$, the value y is also equidistributed in S_3. Obviously, the results of the following tests in step 6, 8 and 9 depend only on y and ϵ_3.

Then $\mathcal{PR}_3 = \mathcal{PR}_{3a} + \mathcal{PR}_{3b}$, where $\mathcal{PR}_{3a}$ and $\mathcal{PR}_{3b}$ are the probabilities that n is declared prime in step 6 and 10, respectively. We have $\mathcal{PR}_{3a} = |S_3|^{-1}$, since the corresponding event occurs only in case $y = 1$.

In the case that $\Phi_3(z)$ has no solution in $G_{1,1}(n,c)$, we have $\mathcal{PR}_{3b} = 0$, so that the proof is finished for this case.

Now let z_0 be such that $\Phi_3(z_0) = 0$ holds, and put $a := \min\{v_3(|C_i|)\}$. $\Phi_3(z_0) = 0$ must hold modulo all prime factors p_i, implying that $\phi_{n,p_i}(z_0)$ is of order 3 for $i = 1, \ldots, \omega$ and hence $a \geq 1$. Since S_3 has ω cyclic factors, there are exactly 2^ω values z_0 with $\Phi_3(z_0) = 0$. Note that the equation $y^{3^a} = z_0$ has no solution in $G_{1,1}(n,c)$, since otherwise all factors C_i would have an order divisible by 3^{a+1}. Hence the set of solutions of $y^{3^i} = z_0$ is a coset of the subgroup of elements of order 3^{a-1} in S_3 and therefore it has cardinality $3^{(a-1)\omega}$. So the probability that $\epsilon'_3 = z_0$ holds in step 7 is $\left(3^{(a-1)\omega}\right)|S_3|^{-1}$ for a fixed z_0 ; and since $|S_3| \geq 3^{a\omega}$, this is at most $3^{-\omega}$.

Counting the number of possible values ϵ'_3 we conclude $\mathcal{PR}_{3b} = 2^\omega \cdot 3^{-\omega}$ from the test in step 8 in case $\epsilon_3 = 1$. In case $\epsilon_3 \neq 1$ we obtain $\mathcal{PR}_{3b} = 2 \cdot 3^{-\omega}$ from the test in step 9. Note that $\mathcal{PR}_{3a} = |S_3|^{-1} \leq 3^{-\omega}$ holds. This proves the formula for $\mathcal{PR}_3$.

Note that the algorithm outputs the value $\epsilon = 1$ only if it stops after step 6, implying $\mathcal{PR}_{31} \leq \mathcal{PR}_{3a} \leq 3^{-\omega}$.
□

Proposition 13 *For the value $\beta_{3,t}(n,c)$ we have:*

$$\beta_{3,t}(n,c) \leq 2^{|\mathbb{P}(n)|-1} \cdot (\beta_3(n,c,3))^t \quad ,$$

where $\beta_3(n,c,\kappa)$ is given by:

$$24^{1-|\mathbb{P}(n)|} \cdot \prod_{\substack{p \in \mathbb{P}(n), \\ (c/p)=-1}} p^{2-2v_p(n)} \frac{(n/p-1, (p^2-1)/\kappa)}{(p^2-1)/\kappa} \cdot \prod_{\substack{p \in \mathbb{P}(n), \\ (c/p)=1}} p^{2-2v_p(n)} \frac{\kappa(n^2/p^2-1, p-1)}{(p-1)^2} \quad .$$

One of the following two improvements is possible (but not both): Either the factor $2^{|\mathbb{P}(n)|-1}$ in the expression for $\beta_{3,t}(n,c)$ may be replaced by 3^{-t} or κ may be set to one.

Proof

Let $\beta_1(n,c)$ and $\beta_{1,t}(n,c)$ be as defined in the last section and assume that n is composite, but declared prime by Algorithm SQFT3.

Case 1: $\Phi_3(z) = 0$ has no solution in $R(n,c)$.
Then the input ϵ_3 for Algorithm SQFT3$_{\text{round}}$ is always one and $\mathcal{PR}_3 = |S_3|^{-1}$ holds by Lemma 12. Hence $\beta_{3,t}(n,c) \le \beta_{1,1}(n,c)^t \cdot |S_3|^{-t} \le \beta_1(n,c)^t \cdot |S_3|^{-t}$ by Proposition 4. From Proposition 4 and Lemma 12 we obtain:

$$\begin{aligned}\beta_1(n,c)\cdot|S_3|^{-1} \quad &\le \quad \frac{1}{3}\cdot 24^{1-|\mathbb{P}(n)|}\cdot \prod_{\substack{p\in\mathbb{P}(n),\\(c/p)=-1}} p^{2-2v_p(n)}\frac{(n/p-1,p^2-1)}{3^{v_3(n/p-1\,,p^2-1)-1}\cdot(p^2-1)}\\ \cdot \prod_{\substack{p\in\mathbb{P}(n),\\(c/p)=1}} p^{2-2v_p(n)}\frac{3(n^2/p^2-1,p-1)}{(p-1)^2} \quad &\le \quad \frac{1}{3}\cdot\beta_3(n,c,3) \quad .\end{aligned}$$

Here we have written $v_3(a,b)$ for $v_3(\gcd(a,b))$. To check the last inequality, note that in case $v_3(n/p-1,p^2-1)=0$ we have $(n/p-1,p^2-1) = (n/p-1,(p^2-1)/3)$, since $3|(p^2-1)$ for all $p \neq 3$.

Case 2: $\Phi_3(z) = 0$ has a solution in $R(n,c)$.
First, note that we have:

$$\beta_3(n,c,3) \;\ge\; \beta_3(n,c,1) \;\ge\; 3^{1-|\mathbb{P}(n)|}\beta_1(n,c) \;\ge\; 3^{1-|\mathbb{P}(n)|}\beta_{1,1}(n,c)\;.$$

We will show:

$$\beta_{3,t}(n,c) \le 2^{|\mathbb{P}(n)|-1}\cdot 3^{t(1-|\mathbb{P}(n)|)}\cdot(\beta_{1,1}(n,c))^t\;,$$

by in induction over t. For $t=1$ this follows from $\beta_{3,1}(n,c) \le \beta_{1,1}(n,c)\cdot\mathcal{PR}_3(1)$ and Lemma 12. If the algorithm computes an $\epsilon'_3 \neq 1$ after the first test, then the probability $\mathcal{PR}_X$ that the algorithm declares n composite in any of the following $t-1$ tests is at most:

$$\mathcal{PR}_X \le (\beta(n,c))^{t-1}\cdot 3^{(t-1)(1-|\mathbb{P}(n)|)} \quad ,$$

by Lemma 12. So we have:

$$\begin{aligned}\beta_{3,t}(n,c) &\le \beta_{1,1}(n,c)\cdot(\mathcal{PR}_{31}\cdot\beta_{3,t-1}(n,c)+(\mathcal{PR}_3(1)-\mathcal{PR}_{31})\cdot\mathcal{PR}_X)\\ &\le (\beta_{1,1}(n,c))^t\cdot 3^{(t-1)(1-|\mathbb{P}(n)|)}\cdot\left(3^{-|\mathbb{P}(n)|}\cdot 2^{|\mathbb{P}(n)|-1}+2^{|\mathbb{P}(n)|}\cdot 3^{-|\mathbb{P}(n)|}\right)\\ &= (\beta_{1,1}(n,c))^t\cdot 2^{|\mathbb{P}(n)|-1}\cdot 3^{t(1-|\mathbb{P}(n)|)} \quad .\end{aligned}$$

□

5 Implementation

In this section we estimate the run time of Algorithm SQFT3. The main ideas in this section are similar to the corresponding estimations for Algorithm EQFTac in [4]. We obtain a bound of $2\log_2 n(1+o(1))$ for one test round. Since Algorithm SQFT is a simplified version of Algorithm SQFT3, the same bound also holds for Algorithm SQFT.

Algorithm EQFTac in [4] achieves the same bound per test round except in cases where n^2-1 is divisible by a high power of 2 or 3.

5.1 Run Time Estimations for Algorithms SQFT and SQFT3

We represent $z \in R(n,c)$ in the usual way by $z = A_z x + B_z$, with $A_z, B_z \in \mathbb{Z}_n$. Then we have:

Lemma 14 *Let $w, z \in R(n,c)$. Then $w \cdot z$ can be computed with 3 multiplications and $O(\log c)$ additions in $\mathbb{Z}_n$. In case $w = z$ one multiplication can be saved.*

Proof

This is Lemma 4 in [4]. Let $w = A_w x + B_w$, $z = A_z x + B_z$. The basic idea is:

$$z \cdot w = (m_1 + m_2)x + (cA_z + B_z)(A_w + B_w) - cm_1 - m_2 \ ,$$

with $m_1 = A_z B_w$, $m_2 = B_z A_w$. Note that $m_1 = m_2$ in case $w = z$.
□

We also take the following lemma from [4].

Lemma 15 *Let n be an odd composite number that is not a perfect square. Let $\pi_-(x,n)$ denote the number of primes $p \leq x$ with $(p/n) = -1$, and let $\pi(x)$ be the total number of primes $p \leq x$. Assuming the extended Riemann Hypothesis (ERH), there exists a constant c_1 (independent of n) such that:*

$$\frac{\pi_-(x,n)}{\pi(x)} > \frac{1}{3} \quad \textit{for all} \quad x \geq c_1(\log n \log\log n)^2 \ .$$

Assuming ERH, Lemma 15 states that we can find a "small" nonresidue c of size $O((\log n \log\log n)^2)$ in Algorithm MR2 with $(c/n) = -1$, with an expected expense of calculating three Jacobi symbols.

In the sequel we frequently have to compute powers of values $z \in R(n,c)$. Here we use the well-known fact that z^a can be computed with $\lceil \log_2 a \rceil$ squarings and $o(\log_2 a)$ multiplications.

We will show a bound for the run time of Algorithm $\text{SQFT3}_{\text{round}}$ in the following proposition. We need another lemma to prove this proposotion.

Lemma 16 *Let R be a ring, $z \in R$, and let $q \in \mathbb{N}$ be constant. For any $a, b \in \mathbb{N}$, we can compute $z^{\lfloor ab/q \rfloor}$ with $log_2 a + O(1)$ squarings and $o(log_2(a))$ multiplications, if $z^{\lfloor a/q \rfloor}$ and $z^{\lfloor b/q \rfloor}$ have been precomputed.*

Proof

Let $\epsilon_a = a \bmod q, \epsilon_b = b \bmod q$ and $\epsilon_{ab} = ab \bmod q$ be such that $0 \leq \epsilon_a, \epsilon_b, \epsilon_{ab} < q$ holds. Then we have:

$$z^{\lfloor ab/q \rfloor} = z^{(ab-\epsilon_{ab})/q} = (z^{(b-\epsilon_b)/q})^a (z^{(a-\epsilon_a)/q})^{\epsilon_b} \cdot z^{(\epsilon_a \epsilon_b - \epsilon_{ab})/q} = (z^{\lfloor b/q \rfloor})^a (z^{\lfloor a/q \rfloor})^{\epsilon_b} \cdot z^{(\epsilon_a \epsilon_b - \epsilon_{ab})/q} \ .$$

Now the lemma follows, since $0 \leq \epsilon_a \epsilon_b - \epsilon_{ab} < q^2$, $0 \leq \epsilon_b < q$, and q is constant.
□

Proposition 17 *On input $z \in R(n,c)$ we can do the following computations simultaneously with $\log_2 n(1+o(1))$ squarings, $o(\log_2 n)$ multiplications and $O(1)$ inversions in $R(n,c)$:*

- *Computing the value z^n .*
- *Checking that z is invertible in $R(n,c)$ and that $\bar{z} = z^n$ holds.*
  If any of these two checks fails, we will say that the input z is bad.
- *Computing the value $z^{(n^2-1)/8}$, under the condition that z is not bad.*
- *Checking if $z^r = 1$ holds, with r given by $n^2 - 1 = 3^u r$, $u = v_3(n^2-1)$,*
  under the condition that z is not bad.
- *Computing the value $z^{3^{i-1}r}$, with $i := \min\{j : 1 \le j \le u,\ z^{3^j r} = 1\}$, or*
  proving that this minimum does not exist, under the condition that z is not bad.

Proof

Since $3^u | (n^2-1)$ and $u > 0$, we can easily find integers e, f and g with $u = ef + g$; $0 \le g \le f$; $e, f > 0$ and $e, f, g = O(\sqrt{log_3 n})$. We treat the case $n = 2 \bmod 3$ in detail, and we only give the basic idea for the simpler case $n = 1 \bmod 3$.

Case $n = 2 \bmod 3$

Then we have $n + 1 = 3^u \cdot s$ for the integer $s = r/(n-1)$. We first compute the sequence

$$t_0 = z^{\lfloor s/8 \rfloor},\ t_1 = z^{\lfloor 3^g \cdot s/8 \rfloor},\ t_2 = z^{\lfloor 3^{f+g} \cdot s/8 \rfloor},\ t_3 = z^{\lfloor 3^{2f+g} \cdot s/8 \rfloor}, \ldots,$$
$$t_e = z^{\lfloor 3^{(e-1)f+g} \cdot s/8 \rfloor},\ t_{e+1} = z^{\lfloor 3^{ef+g} \cdot s/8 \rfloor} = z^{\lfloor (n+1)/8 \rfloor}\ .$$

Given t_{j-1} and z^{3^f}, the value t_j can be computed with $\log_2 3^f + O(1)$ squarings and $o(\log_2 3^f)$ multiplications for $j > 1$ by Lemma 16. Similarly, we need $\log_2 3^g + O(1)$ squarings and $o(\log_2 3^g)$ multiplications for computing t_1 if t_0 and z^{3^g} are given. We further need $\log_2 s + O(1)$ squarings and $o(\log_2 s)$ multiplications for computing t_0. Precomputation of z^{3^f}, z^{3^g} costs $O(f)$ multiplications and squarings, since $g \le f$. Thus the whole sequence $t_0 \ldots t_e$ can be computed with $\log_2 n + O(e+f)$ squarings and $o(\log_2 n) + O(e+f)$ multiplications.

Next we also compute z^{-1}. This costs one inversion in $R(n,c)^*$. Now from $t_{e+1} = z^{\lfloor (n+1)/8 \rfloor} = z^{(n+1-\epsilon)/8}$ (for some integer $0 \le \epsilon < 8$) and z^{-1} we can compute z^n and check if $\bar{z} = z^n$ holds. This costs $O(1)$ additional squarings and multiplications.

Next we compute $z^{(n^2-1)/8}$ with $O(1)$ multiplications, squarings and inversions as follows:

$$z^{(n^2-1)/8} = z^{n(n+1-\epsilon)/8} z^{(\epsilon-1)(n+1-\epsilon)/8} z^{\epsilon(\epsilon-2)/8} = \overline{t_{e+1}} (t_{e+1})^{\epsilon-1} z^{\epsilon(\epsilon-2)/8}\ .$$

In order to check if $z^r = 1$ holds, we may check $\overline{z^s} = z^s$ instead. This is equivalent to $z^r = 1$, since $\overline{z^s} = z^{ns} = z^{r+s}$ and z is invertible. This costs $O(1)$ more multiplications and squarings.

Finally, we have to compute $z^{3^{i-1}r}$ with $i := \min\{j : 1 \le j \le u\,,\, z^{3^j r} = 1\}$. We assume that this minimum exists. Otherwise the calculations below prove the non-existence of this minimum. Here we use a backtracing mechanism similar to the one used in the original QFT in [6].

Define $\tau(j) = \max\{0, (j-1)f + g\}$. Then $t_j = z^{\lfloor 3^{\tau(j)}s/8 \rfloor}$. Given $t_j, j = 0, \ldots, e+1$, we can easily compute the quantities $t'_j = z^{3^{\tau(j)}s}, j = 0, \ldots, e+1$. Altogether, it costs $O(e)$ additional squarings and multiplications to compute $t'_0, \ldots, t'_{e+1}$. Now $z^{3^{\tau(j)}r} = 1$ is equivalent to $\overline{z^{3^{\tau(j)}s}} = z^{3^{\tau(j)}s}$, since $\bar{z} = z^n$, $r = s(n-1)$, and z is invertible. So we may find $J := \min\{j : 1 \le j \le e+1\,,\, z^{3^{\tau(j)}r} = 1\}$ with $O(e)$ squarings and multiplications. The value $i-1$ must satisfy $\tau(J-1) \le i-1 \le \tau(J) - 1$. Now we compute

$$z^{3^{\tau(J-1)}r} = z^{3^{\tau(J-1)}(ns-s)} = \overline{z^{3^{\tau(J-1)}s}} z^{-3^{\tau(J-1)}s} = \overline{t'_{J-1}}(t'_{J-1})^{-1}.$$

This costs $O(1)$ multiplications and inversions.

From the above considerations we know that $z^{3^{i-1}r}$ must be a member of the sequence

$$z^{3^{\tau(J-1)}r}, z^{3\cdot 3^{\tau(J-1)}r}, z^{3^2\cdot 3^{\tau(J-1)}r}, \ldots, z^{3^{f-1}\cdot 3^{\tau(J-1)}r}\,.$$

But this sequence can be calculated with $O(f)$ additional squarings and multiplications.

Altogether we need $\log_2 n + O(e+f) = \log_2 n(1+o(1))$ squarings, $o(\log_2 n) + O(e+f) = o(\log_2 n)$ multiplications, and O(1) inversions.

Case $n = 1 \bmod 3$

Then we have $n - 1 = 3^u \cdot s$ for the integer $s = r/(n+1)$. We first compute the sequence

$$t_0 = z^{\lfloor s/8 \rfloor},\ t_1 = z^{\lfloor 3^g \cdot s/8 \rfloor},\ t_2 = z^{\lfloor 3^{f+g} \cdot s/8 \rfloor},\ t_3 = z^{\lfloor 3^{2f+g} \cdot s/8 \rfloor}, \ldots,$$
$$t_e = z^{\lfloor 3^{(e-1)f+g} \cdot s/8 \rfloor},\ t_{e+1} = z^{\lfloor 3^{ef+g} \cdot s/8 \rfloor} = z^{\lfloor (n-1)/8 \rfloor}\,.$$

This can be done with $n + o(n)$ squarings and $o(n)$ multiplications as in the case $n = 2 \bmod 3$.

Now from $t_{e+1} = z^{\lfloor (n-1)/8 \rfloor} = z^{(n-1-\epsilon)/8}$ (for some integer $0 \le \epsilon < 8$) we can compute z^n and check if $\bar{z} = z^n$ holds. This costs $O(1)$ additional squarings and multiplications.

Next we compute $z^{(n^2-1)/8}$ with $O(1)$ multiplications and squarings as follows:

$$z^{(n^2-1)/8} = z^{(n+1+\epsilon)(n-1-\epsilon)/8} z^{\epsilon(\epsilon+2)/8} = N(t_{e+1})(t_{e+1})^{\epsilon} z^{\epsilon(\epsilon+2)/8}\,.$$

Define $\tau(j) = \max\{0, (j-1)f + g\}$. Note that $z^{3^{\tau(j)}r} = z^{3^{\tau(j)}(n+1)s} = N(z^{3^{\tau(j)}s})$. So we can compute $z^{3^{\tau(j)}r}; j = 0, \ldots, e+1$ from $t_j = z^{\lfloor 3^{\tau(j)}s/8 \rfloor}$ with $O(e)$ squarings and multiplications. Given this sequence, we can compute $z^{3^{i-1}r}$ with $i = \min\{j : 1 \le j \le u\,,\, z^{3^j r} = 1\}$ with $O(f)$ squarings and multiplications. Here we use the same backtracing mechanism as in case $n = 2 \bmod 3$.

Altogether we need $\log_2 n(1+o(1))$ squarings, $o(\log_2 n)$ multiplications, and no inversion in this case.

□

Theorem 18 *Assuming ERH, there are implementations of Algorithms SQFT and SQFT3 that have an expected run time equivalent to* $(2t+1)\log_2 n(1+o(1))$ *multiplications in* $\mathbb{Z}_n$. *Here* t *is the number of test rounds.*

Proof

It suffices to show the theorem for Algorithm SQFT3, since Algorithm SQFT is faster than Algorithm SQFT3. Algorithm SQFT3 consists of one application of Algorithm MR2 and t applications of Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$.

In step 1 of Algorithm SQFT3, a fixed number of trial divisions is performed. This can be done at cost equivalent to $O(1)$ multiplications in in $\mathbb{Z}_n$. In step 2, Algorithm MR2 is called. There the following calculations must be done:

- Possibly, the computation of $\sqrt{n}$. This costs $O(\log\log n)$ multiplications.
- Possibly, the computation of Jacobi symbols (c/n) for random integers $2 < c \leq c_1(\log n \log\log n)^2$. By Lemma 15, the expected number of such computations is ≤ 3. This can be done at cost equivalent to $O(\log\log n)$ multiplications in $\mathbb{Z}_n$.
- Essentially, a Miller-Rabin test with basis 2 or c. This costs $\log_2 n + O(1)$ squarings and $o(\log n)$ multiplications in $\mathbb{Z}_n$.
- The computation of the output value ϵ of Algorithm MR2. This costs O(1) more additions and multiplications in $\mathbb{Z}_n$.

Thus the cost for steps 1 to 4 in Algorithm SQFT3 can be bounded by the expense corresponding to $\log n(1+o(1))$ multiplications in $\mathbb{Z}_n$.

In the final step 5 of Algorithm SQFT3 we apply Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$ t times. In each round we must select a random $z \in R(n,c) \setminus \{0\}$ with $(N(z)/n) \neq 1$. (In case $(N(z)/n) = 0$ we obtain a factorisation of n). We can do this with an expected run time equivalent to poly$(\log\log n)$ multiplications in $\mathbb{Z}_n$, corresponding to $O(1)$ calculations of Jacobi symbols on numbers of size n. Note that the other activities performed by Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$ are just the activities enumerated in Proposition 17. So the bound in that proposition applies for Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$, and using Lemma 14 we can bound the expense for that algorithm by $\log n(2+o(1))$ multiplications in $\mathbb{Z}_n$. Note that the size of c in $R(n,c)$ is bounded by Lemma 15.

□

5.2 Optimisations

Clearly, the implementation of Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$ sketched in the proof of proposition 17 is not optimal, as far as space is concerned. There is also room for improvement in speed; although it is unlikely that the bound for the run time can be much improved. Note that Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ can be implemented without inversions in $R(n,c)^*$.

Next we remark some ideas for optimisations of Algorithms SQFT and SQFT3.

In Step 5 of Algorithm SQFT, we call Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ t times, once for each test round. We can modify Algorithm SQFT as follows. Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ is called

only in the first test round, and in the subsequent $t-1$ test rounds we may call the following Algorithm $\mathrm{SQFT}_{\mathrm{subs}}$ instead:

Algorithm $\mathrm{SQFT}_{\mathrm{subs}}$

`Input` Same as for Algorithm $\mathrm{SQFT}_{\mathrm{round}}$.
`Output` Same as in Algorithm $\mathrm{SQFT}_{\mathrm{round}}$.
[1] Select a random $z \in R(n,c)^*$.
[2] `If` $z^n \neq \bar{z}$ `then output` "n is composite" and `stop` .
[3] `If` $z^{\frac{n^2-1}{8}} \notin \{\pm 1, \pm\epsilon, \pm\epsilon^2, \pm\epsilon^3\}$ `then output` "n is composite" and `stop` .
[4] `output` "n is possibly prime" and `stop` .

This saves $t-1$ computations of Jacobi symbols. A similar optimisation can be applied to Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$. The computation of a Jacobi symbol costs essentially one modular inversion in $\mathbb{Z}_n^*$, which is much less than the cost for a modular exponentiation. However, on smart cards it may well be worth saving modular inversions for two reasons. First, some smart cards have special hardware support for modular multiplication, but not for modular inversion. On the other hand, countermeasures against hardware attacks may be more expensive for modular inversion than for modular multiplication.

Proposition 4 also holds for the modified Algorithm SQFT. The reason for this is as follows: We must find a $z' \in G_{8,\epsilon}(n,c) \subset G_{1,1}(n,c)$, so that we may apply Lemma 10 in the proof of Proposition 4. In other words, the existence of such a z' asserts that all factors of the 2-Sylow groups of $R(n,c)^*$ have an order divisible by $2^{v_2(n^2-1)}$. The simplest way to establish the existence of such a z' is to start with a z satisfying $(N(z)/n) = -1$ in Algorithm $\mathrm{SQFT}_{\mathrm{round}}$. Once that existence has been established (or n has been proved composite), we no longer need to deal with Jacobi symbols. So we may replace Algorithm $\mathrm{SQFT}_{\mathrm{round}}$ by Algorithm $\mathrm{SQFT}_{\mathrm{subs}}$ in subsequent test rounds. A similar idea has also been used in Algorithm EFTCwc in [4].

The most cumbersome part of Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$ is the computation of the minimum i in step 5 in cases where $v_3(n^2-1)$ is large. The bulk of the proof of Proposition 17 deals with the estimation of the expense for the computation of this minimum. Although this computation can be done quite efficiently for large n, the overhead may be considerable if n has moderate size, as in cryptographic applications. This is most noteworthy in case of prime n, since composite n are already ruled out by the preceding application of Algorithm MR2 with high probability. If the value z selected in step 1 of Algorithm $\mathrm{SQFT3}_{\mathrm{round}}$ satisfies $\Phi_3(z^{(n^2-1)/3}) = 0$, (implying that the minimum i computed in step 5 equals $v_3(n^2-1)$), then we can omit the computation of the minimum i in subsequent test rounds; and we can check if $z^{(n^2-1)/3} \in \{1, \epsilon_3^{\pm 1}\}$ holds in subsequent iterations instead. The reason for this is that all factors of the 3-Sylow group of $R(n,c)^*$ have an order divisible by $3^{v_3(n^2-1)}$, if any such z exists. For a random $z \in R(n,c)^*$ and prime n, condition $\Phi_3(z^{(n^2-1)/3}) = 0$ is satisfied with probability $1/3$, since $R(n,c)^*$ is a cyclic group of order n^2-1. This is essentially the

same idea as above, with the 2-Sylow groups replaced by 3-Sylow groups. Using more sophisticated variants of this idea, we can avoid the computation of i in subsequent iterations of step 5 with even higher probability.

6 Worst Case Behaviour

In this section we will show the following worst-case behaviour for Algorithms SQFT and SQFT3:

Proposition 19 *The probability $\beta_{1,t}(n)$ that Algorithm SQFT declares a composite number n prime after t rounds is at most 2^{-12t}.*

Proposition 20 *The probability $\beta_{3,t}(n)$ that algorithm SQFT3 declares a composite number n prime after t rounds is at most:*

$$2^4 \cdot 24^{-4t} \approx 2^{4-18.36t} .$$

We give the details for the proof of Proposition 20; and we only sketch the (simpler) proof of Proposition 19.

Let $\omega := |\mathbb{P}(n)|$ be the number of different prime factors of n. Let $\Omega := \sum_{p\in\mathbb{P}(n)} v_p(n)$ be the number of prime factors, with counting multiplicity.

Proof of Proposition 20

Proposition 13 yields the following bound for $\beta_{3,t}(n)$:

$$\begin{aligned}\beta_{3,t}(n) &\leq \max_{(c/n)=-1} 2^{\omega-1}\cdot 24^{t(1-\omega)} \cdot \prod_{p\in\mathbb{P}(n)} p^{t(2-2v_p(n))} \cdot \prod_{\substack{p\in\mathbb{P}(n),\\(c/p)=1}} \frac{3^t}{(p-1)^t} \\ &\leq \max_{(c/n)=-1} 2^{\omega-1}\cdot 24^{t(1+2\omega-3\Omega-\nu)}\,;\ \text{with } \nu = |\{p\in\mathbb{P}(n) : (c/p)=1\}| . \quad (4)\end{aligned}$$

The last inequality holds, since all prime factors of n satisfy $p > 200 > 24^{3/2}$. Since $(c/n) = -1$ and the Jacobi symbol is multiplicative, ν cannot be zero if Ω is even. This proves the proposition for all cases with $\Omega \geq 4$ and also for all cases with $\Omega > \omega \geq 2$. Note that in case $\Omega = \omega$ (i.e. n is not divisible by a square of a prime) we also have $\omega = \nu + 1 \pmod 2$, since $(c/n) = -1$.

So we are left with the following cases:

Case 1: $n = p_1p_2$**.**

We may assume $(c/p_1) = -1, (c/p_2) = 1$. Then from Proposition 13 we obtain:

$$\beta_{3,t}(n,c) \leq 2\cdot 3^t\cdot 24^{-t}\cdot \frac{\gcd(p_1^2-1, p_2-1)^t}{(p_1^2-1)^t(p_2-1)^t} \leq \frac{2\cdot 3^t\cdot 24^{-t}}{(p_1^2-1)^t} \leq 2\cdot 24^{-4t}\,;\text{for } p_1 \geq 211 .$$

(Here the reader should check both possible improvements in Proposition 13 separately!)

Case 2: $n = p_1p_2p_3$**.**

Then either $(c/p_i) = 1$ holds for exactly two of the primes p_1, p_2, p_3 or for none of them. In the first case, the proposition follows from (4). So we may assume $200 < p_1 < p_2 < p_3$ and $(c/p_i) = -1$ for $i = 1, 2, 3$. Then from Proposition 13 we obtain:

$$\begin{aligned}\beta_{3,t}(n,c) &\leq 2 \cdot 3^{2t} \cdot 24^{-2t} \cdot \prod_{i=1}^{3} \left(\frac{\gcd(n/p_i - 1, p_i^2 - 1)}{p_i^2 - 1} \right)^t \\ &\leq 2 \cdot 3^{2t} \cdot 24^{-2t} \cdot \frac{1}{(p_1^2 - 1)^t} \; ; \quad \text{by Lemma 24 in section 9} \\ &\leq 2 \cdot 24^{-4t} \; ; \quad \text{for } p_1 > 200 \; .\end{aligned}$$

(Here the reader should check both possible improvements in Proposition 13 separately!)

Case 3: n **is a prime power** p^m**.**

Since perfect squares are declared composite by Algorithm MR2, we have $m \geq 3$ and therefore the proposition follows from (4).

□

Sketch proof of Proposition 19

From Proposition 4 and the fact that $p \geq 23 > 8^{3/2}$ holds for all prime factors of n, we obtain:

$$\beta_{1,t}(n) \leq \max_{(c/n)=-1} 2^{\omega-1} \cdot 8^{t(1+2\omega-3\Omega-\nu)} \text{; with } \nu = |\{p \in \mathbb{P}(n) : (c/p) = 1\}| \; .$$

Now all cases can be treated similar to those in the proof of Proposition 20. For the case $n = p_1p_2, (c/p_1) = -1, (c/p_2) = 1$, we obtain:

$$\beta_{1,t}(n,c) \leq 8^{-t} \cdot \frac{\gcd(p_1^2 - 1, p_2 - 1)^t}{(p_1^2 - 1)^t (p_2 - 1)^t} \leq \frac{8^{-t}}{(p_1^2 - 1)^t} \leq 2^{-12t} \quad \text{; for } p_1 \geq 23 \; .$$

□

7 Average Case Behaviour

We analyse the average case behaviour of Algorithm SQFT in this section.

Let M_k be the set of odd $k-$bit integers, i.e. $M_k = \{n \in \mathbb{N} : 2^{k-1} < n < 2^k, n \text{ odd}\}$. We consider an algorithm that repeatedly chooses a random number from M_k and applies Algorithm SQFT with t test rounds to that number, until a number is found that is declared prime by Algorithm SQFT. We are interested in the error probability $q_{k,t}$ that the above algorithm (with t tests) returns a composite number in M_k.

Both, the error probabilities calculated for Algorithm SQFT and the error probabilities calculated for Algorithm EQFTac in [4] are also valid for Algorithm SQFT3. The reason for this is that the bound 5 for the error probability of Algorithm SQFT as well as the corresponding bound for Algorithm EQFTac in [4] are also valid for Algorithm SQFT3.

Let $\Omega = \Omega(n)$ be defined by $\Omega(n) := \sum_{p \in \mathbb{P}(n)} v_p(n)$ as in the last section. Furthermore, let p be the largest prime factor of n.

We use the following bound for the probability $\beta_{1,t}(n)$ that Algorithm SQFT declares a composite number n prime:

$$\beta_{1,t}(n) \;\leq\; \beta_1(n)^t \;, \tag{5}$$

with $\beta_1(n)$ being defined by:

$$\beta_1(n) \;=\; 8^{1-\Omega(n)} \;\cdot \max\left\{ \frac{(n/p-1, p^2-1)}{p^2-1}, \frac{1}{p-1} \right\} \;. \tag{6}$$

This bound follows from Proposition 4. In addition to (6), we put $\beta_1(n) = 0$ if n is divisible by any prime $p < 23$. Since prime factors $p < 23$ are ruled out by trial division, (5) is still valid with this definition.

We define C_m to be the set of odd composite natural numbers with $(\beta_{1,t})^{1/t} > 2^{-m}$ for all t. Obviously, every $n \in C_m$ satisfies $\beta_1(n) > 2^{-m}$. Proposition 19 implies that C_m is empty for $m \leq 12$.

Our main goal is to estimate $|C_m \cap M_k|$. As in [3], [5] and [4] such an estimate allows us to give effective bounds for the error probability of the above algorithm. In the next section we also use this estimate to obtain a bound for the error probability of the incremental search version of the above algorithm, using the same techniques as in [1].

We define $N(m,k,j)$ to be the set of integers $n \in C_m \cap M_k$ with $\Omega(n) = j$.

Since (5) and (6) imply $(\beta_{1,t}(n))^{1/t} \leq \beta_1(n) \leq 2^{3(1-\Omega(n))}$ we have $\Omega(n) \leq m/3 + 1$ for any $n \in C_m$. So we have:

$$|C_m \cap M_k| = \begin{cases} \sum_{2 \leq j \leq m/3+1} |N(m,k,j)| & \text{for} \quad m > 12 \;; \\ 0 & \text{for} \quad m \leq 12 \;. \end{cases} \tag{7}$$

Since any $n \in N(m,k,j)$ has j prime factors (counting multiplicity) and is greater than 2^{k-1}, the largest prime factor p of n satisfies $p > 2^{(k-1)/j}$.

Now we will estimate $|N(m,k,j)|$. For our estimation we assume:

$$m \leq \sqrt{12(k-1)} - 4 \;. \tag{8}$$

Then for any $j > 0$ we have:

$$m + 4 \leq \sqrt{12(k-1)} \leq 3j + (k-1)/j \;. \tag{9}$$

For the largest prime factor p of $n \in N(m,k,j)$ we have:

$$\begin{aligned} \frac{1}{p-1} \;&<\; \frac{2}{p} \\ &<\; 2^{1-(k-1)/j} \quad ; \text{ since } \quad p > 2^{(k-1)/j} \\ &\leq\; 2^{-m-3(1-j)} \quad ; \text{ by (9)} \\ &<\; \max\left\{ \frac{(n/p-1, p^2-1)}{p^2-1}, \frac{1}{p-1} \right\} \quad . \end{aligned} \tag{10}$$

The last inequality follows from (6), since $\Omega = j$ and $\beta_1(n) > 2^{-m}$. We define:

$$d(p,n) \;=\; \frac{p^2-1}{(n/p-1,\,p^2-1)}\;, \tag{11}$$

and we put $d = d(p,n)$. Then we have $d < 2^{m+3(1-j)}$ by (10). Therefore, we obtain an upper bound for $|N(m,k,j)|$ as follows. For any prime p with $p > 2^{(k-1)/j}$ and integer d with $d < 2^{m+3(1-j)}$ we count the number of $n \in M_k$ with the property that $p|n, d = d(p,n)$ and $n \neq p$. For each pair (d,p), this is at most the number of solutions to the system

$$n = 0 \bmod p\,,\; n = p \bmod \frac{p^2-1}{d}\;,\; n \neq p \quad .$$

By the Chinese remainder theorem, the number of solutions is at most

$$\frac{2^k d}{p(p^2-1)} \quad .$$

Thus, counting the number of $n \in M_k$ with $p|n, d = d(p,n)$, $n \neq p$ we obtain:

$$|N(m,k,j)| \;\leq\; \sum_{p \geq 2^{(k-1)/j}} \sum_{\substack{d \leq 2^{m+3(1-j)} \\ d|(p^2-1)}} \frac{2^k d}{p(p^2-1)}\;.$$

Changing the summation order and using $p \geq 23$, i.e. $p^2/(p^2-1) \leq \frac{529}{528}$, we obtain:

$$|N(m,k,j)| \;\leq\; 2^k \cdot \frac{529}{528} \cdot \sum_{d \leq 2^{m+3(1-j)}} \sum_{\substack{p \geq 2^{(k-1)/j} \\ p^2 = 1 \bmod d}} \frac{d}{p^3}\;. \tag{12}$$

We proceed as in [4] by estimating the inner sum. Let $T(d)$ be the number of integer solutions $0 < x \leq d$ satisfying $x^2 = 1 \pmod d$. Then we have:

$$\begin{aligned}
\sum_{\substack{p \geq 2^{(k-1)/j} \\ p^2 = 1 \bmod d}} \frac{d}{p^3} \;&\leq\; T(d) \sum_{u=0}^{\infty} \frac{d}{(du + 2^{(k-1)/j})^3} \\
&=\; \frac{T(d)}{d^2} \sum_{u=0}^{\infty} (u + \frac{2^{(k-1)/j}}{d})^{-3} \\
&\leq\; 0.82 \cdot T(d) \cdot 2^{-2(k-1)/j} \quad ;\text{ by Lemma 26, since } 2^{(k-1)/j} > 2d.
\end{aligned}$$

To check the last inequality, note that $d \leq 2^{m+3(1-j)}$ holds in (12) and hence $2^{(k-1)/j} \cdot d^{-1} \geq 2$ by (9). Now from (12) and Lemma 25 we obtain the following:

$$\begin{aligned}
|N(m,k,j)| \;&\leq\; 2^k \cdot \frac{529}{528} \cdot 0.82 \cdot 2^{-2(k-1)/j} \sum_{d \leq 2^{m+3(1-j)}} T(d) \\
&\leq\; \frac{529}{528} \cdot 0.82 \cdot 1.205 \cdot 2^k \cdot 2^{5m/4+15/4-15j/4-2(k-1)/j} \\
&<\; 2^{k+5m/4+15/4-15j/4-2(k-1)/j} \quad .
\end{aligned} \tag{13}$$

From (7) and (13) we immediately obtain:

Table 2: Lower bounds for $-\log_2 q_{k,t}$

$k \setminus t$	1	2	3	4	5
300	48	100	124	143	160
400	57	118	146	169	189
500	65	134	166	192	214
600	72	148	184	212	237
1000	96	197	243	281	314

Proposition 21 *We have $C_m = \emptyset$ for $m \leq 12$; and for positive integers m, k with $12 < m \leq \sqrt{12(k-1)} - 4$ we have:*

$$|C_m \cap M_k| \leq 2^{k+5m/4+15/4} \sum_{2 \leq j \leq m/3+1} 2^{-15j/4-2(k-1)/j} \quad .$$

Now, using exactly the same argument as in Proposition 1 of [5], we obtain

$$q_{k,t} \leq \frac{2^{-Mt}|M_k \setminus C_m| + \sum_{m=13}^{M} 2^{-(m-1)t}|M_k \cap C_m|}{\pi(2^k) - \pi(2^{k-1})} \quad , \tag{14}$$

where M is a free parameter to be chosen in the range $13 \leq M \leq \sqrt{12(k-1)} - 4$. If no such M exists, we still have $q_{k,t} \leq 2^{-12t}|M_k| \cdot (\pi(2^k) - \pi(2^{k-1}))^{-1}$. Proposition 2 in [5] states

$$\pi(2^k) - \pi(2^{k-1}) \geq 0.71867 \cdot 2^k/k \quad . \tag{15}$$

This allows us to obtain numerical estimates for $q_{k,t}$ from (14), (15) and Proposition 21. Some sample results are shown in Table 2, which contains the negative binary logarithm of the estimates; so that we claim e.g. $q_{500,2} \leq 2^{-134}$.

Comparing our results with those in [4], we see that our results for Algorithm SQFT are slightly better than the corresponding values for Algorithm EQFTac in [4] in case $t = 1$; they are slightly worse in case $t = 2$, and considerably worse in case $t \geq 3$. We have the following explanation for this phenomenon. For large t, Algorithm EQFTac has the advantage of considering also 3^{rd}roots of unity, leading to much better estimates. For $t = 1$, however, Algorithm SQFT has the advantage that it needs not "learn" a 4^{th} or 8^{th} root of unity, since such a root has already been computed by the preceding Miller-Rabin test.

Yet it should be noted that Algorithm SQFT is considerably simpler than Algorithm EQFTac. Note that both algorithms meet the requirements concerning the error probability, as stated e.g. in [7], already after two tests in cases of practical interest, such as $512 \leq k \leq 1024$.

Theorem 17 in [4] states that for $t \geq 2$ the error probability $q_{k,t}^{\text{EQFTac}}$ of Algorithm EQFTac is bounded by

$$O(k^{3/2} 2^{(\sigma_t+1)t} t^{-1/2} 4^{-\sqrt{2\sigma_t tk}})$$

with $\sigma_t = \log_2(24) - 2/t$. Using the same ideas as in [4] we can bound the error probability for Algorithm SQFT with the same expression, where σ_t has to be set to 3 for Algorithm SQFT.

8 Average Case Behaviour on Incremental Search

The algorithms analysed in the last section is seldom used in practice. Usually, we do not want to choose candidates for primes uniformly at random. Instead, one will choose a starting point n_0 in M_k at random and test all values $n_0, n_0 + 2, n_0 + 4, \ldots$ for primality until a probable prime has been found. The reason for doing so is that trial divisions can be combined with techniques like Eratosthenes's sieve to rule out candidates with small prime factors quickly. Another reason for doing so is that on devices with limited computing power, such as smart cards, the generation of random numbers may be expensive. The efficiency of an incremental search algorithm using the Miller-Rabin test for prime number generation is analysed in [2]; its error probability is analysed in [1]. A similar analysis for the extended quadratic Frobenius test has been carried out in [4]. Our error analysis is still valid when composite numbers are ruled out via trial divisions or sieving methods, as long as no prime is ruled out.

For our analysis of the incremental search variant of Algorithm SQFT, we introduce another parameter s, the *sieve length*. This means, when searching for a prime, we first test s candidates $n_0, n_0 + 2, n_0 + 4, \ldots, n_0 + 2(s - 1)$. If none of these candidates turns out to be probably prime, we have to generate a new random start value n_0.

When analysing the incremental search variant of the algorithm SQFT, the error probabilities are considerably higher than those for the random search variant in the last section. This is due to the fact that subsequent tests can no longer be considered as independent. Roughly speaking, the analysis in [1] shows that the error probability is increased by a factor of about s^2, when intervals of length s are considered, and a probable prime is found in the first interval. If many intervals have to be considered, the error probability becomes even higher, see [4] for an analysis of this case. Note that the interval size should be $s = c_2 \cdot \log(2^k)$, for some constant c_2, if we want to find a prime in that interval with probability, say, greater than $1/2$.

There are good reasons to keep the error probability as small as possible also in the case when several intervals have to be searched. One reason is that future versions of standards like [7] might require smaller error probabilities. Another reason is that in practical applications, we might have to find a prime in a much smaller set than M_k. For example, when we search for an RSA key with given public exponent $e = 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$, we will eventually have to reject all primes that are 1 modulo $3, 5, 7, 11,$ or 13.

One way to keep the error probability small is to slowly increase the number of tests performed on each candidate, when no probable prime has been found in a certain number of intervals. Therefore we introduce a new parameter r. If no probable prime has been found after testing all candidates in r intervals, we will increase the number of test rounds in the SQFT by 1. So we arrive at the following algorithm:

Algorithm $\mathbf{SQFT_{inc}}$ *Incremental search version of Algorithm SQFT*

`Input` Bit length k , number t of test rounds, interval size s, increment r.
`Output` A probable prime n in the set M_k, i.e. $2^{k-1} < n < 2^k$.

[1] Put $t' = t$, $i = 0$.
[2] Select a random $n_0 \in M_k$.
[3] `For` $n \in \{n_0, n_0 + 2, n_0 + 4, \ldots, n_0 + 2(s-1)\}$ do the following;
`Call` Algorithm SQFT with input n and number of test rounds t'.
`If` Algorithm SQFT declares n prime `then output` n and `stop` .
[4] Put $i = i + 1$.
[5] `If` $i = 0 \bmod r$ `then put` $t' := t' + 1$.
[6] `Go to` step [2].

In order to analyse Algorithm $\mathrm{SQFT_{inc}}$ we give a bound for its expected run time and for its error probability. We give a rigorous upper bound for the error probability and a heuristic bound for the expected run time. Note that standards like [7] require upper bounds for the error probability, but not for the run time, so that we may be less rigorous when estimating the run time.

Let $Q_{k,t,s,r}$ be the probability that Algorithm $\mathrm{SQFT_{inc}}$ returns a composite number and let $q_{k,t,s}$ be the probability that one execution of the loop in step 3 of the algorithm returns a composite number.

Using exactly the same argument as in [1] we obtain:

Lemma 22 *Let $s = c \cdot \ln(2^k)$ for some constant c. Then for any $M > 12$, we have*

$$\begin{aligned} q_{k,t,s} &\leq s^2 \sum_{i=13}^{M} \frac{|C_m \cap M_k|}{|M_k|} 2^{-t(m-1)} + s \cdot 2^{-tM} \\ &\leq 0.480454(ck)^2 \sum_{i=13}^{M} \frac{|C_m \cap M_k|}{|M_k|} 2^{-t(m-1)} + 0.693148 \cdot ck 2^{-tM} \ . \end{aligned}$$

The lemma is also valid for $M = 12$, if the sum is replaced by 0. Note that C_m is empty for $m \leq 12$ and that the same argument is also used in [4] for algorithm EQFTac. Let $\bar{q}_{k,t,s}$ be the bound for $q_{k,t,s}$ stated in Lemma 22. Obviously, $q_{k,t+t_1,s} \leq 2^{-12t_1} \bar{q}_{k,t,s}$ holds for any $t_1 \geq 0$. Since any error of Algorithm $\mathrm{SQFT_{inc}}$ must occur in one iteration of step 3, we obtain:

$$Q_{k,t,s,r} \leq \bar{q}_{k,t,s}(r + r \cdot 2^{-12} + r \cdot 2^{-2 \cdot 12} + r \cdot 2^{-3 \cdot 12} \ldots) \leq \frac{4096}{4095} r \bar{q}_{k,t,s} \quad .$$

Together with Lemma 22 we obtain:

Table 3: Lower bounds for $-\log_2 Q_{k,t,s,r}$ with $s = c \cdot \ln(2^k)$ and $c = r = 10$.

$k \setminus t$	1	2	3	4	5
300	40	81	105	125	141
400	50	99	127	150	169
500	57	114	146	172	195
600	64	129	164	193	217
1000	88	176	223	261	294

Proposition 23 *Let $s = c \cdot \ln(2^k)$ for some constant c. Then for any $M \geq 12$, we have*

$$Q_{k,t,s,r} \leq 0.5 \cdot r(ck)^2 \sum_{i=13}^{M} \frac{|C_m \cap M_k|}{|M_k|} 2^{-t(m-1)} + 0.7 \cdot rck 2^{-tM} \quad ;$$

(with the sum being empty in case M=12).

Using Proposition 23, we have computed some values $-\log_2 Q_{k,t,s,r}$ with $s = c \cdot \ln(2^k)$ and $c = r = 10$ in Table 3. We have chosen $r = c = 10$ to obtain results comparable to those in [4]. In practice, smaller values for r and c can be chosen.

It remains to bound the run time for Algorithm $\mathrm{SQFT}_{\mathrm{inc}}$. As stated above, we may be less rigorous here than in the analysis of the error probability.

Let $\mathcal{PR}_{\mathrm{FAIL}}(k, s)$ be the probability that the interval $[n_0, \ldots, n_0+2s-2]$, with n_0 chosen uniformly at random in M_k, contains no prime. Assuming Hardy and Littlewood's prime r-tuple conjecture, it is shown in [1] that $\mathcal{PR}_{\mathrm{FAIL}}(k, c \cdot \ln(2^k)) \leq 2\exp(-2c)$ holds for sufficiently large k. E.g. in case $c \geq 1, rc \geq 10$ the probability that the number t' of test rounds in Algorithm $\mathrm{SQFT}_{\mathrm{inc}}$ must ever be increased is at most $3 \cdot 10^{-6}$ and may be neglected.

Practical experience shows that strong pseudoprimes are rare for any basis selected by Algorithm MR2, and that we must test about $\ln(2^k)/2$ candidates until we find a prime number, independent of the interval size s.

Thus the expected run time for Algorithm $\mathrm{SQFT}_{\mathrm{inc}}$ is about

$$(\ln(2^k)/2 + 2t)(1 + c_3)$$

(in units counting the effort for one Miller-Test) for a small constant $c_3 \ll 1$, $c \geq 1$, $rc \geq 10$, and sufficiently large values k. Note that the term depending on k can be improved by using trial divisions and sieving methods, as e.g. stated in [2].

For large k, the above argument for bounding the run time could be made more rigorous by using the prime r-tuple conjecture mentioned above, Lemma 15, and known facts about the distribution of pseudoprimes, see [12].

9 Technical Lemmata

Lemma 24 *Let $n = p_1p_2p_3$ be a product of three different primes p_1, p_2, p_3 with $2 \leq p_1 < p_2 < p_3$. Then the following inequality holds:*

$$\prod_{i=1}^{3} \frac{\gcd(n/p_i - 1, p_i^2 - 1)}{p_i^2 - 1} < \frac{1}{p_1^2 - 1} \; .$$

Proof

This lemma has originally been proved in [6]. For another proof, see [4]. We present a simpler proof here.

Define $z := \prod_{i=1}^{3}(n/p_i - 1)/(p_i^2 - 1)$. Then a straightforward calculation yields:

$$z - 1 \; = \; \left(\sum_{i=1}^{3} \frac{1}{p_i^2 - 1} \right) - \frac{p_1p_2 - 1}{(p_1^2 - 1)(p_2^2 - 1)} - \frac{p_1p_3 - 1}{(p_1^2 - 1)(p_3^2 - 1)} - \frac{p_2p_3 - 1}{(p_2^2 - 1)(p_3^2 - 1)} \; . \quad (16)$$

Since $p_i \neq p_j$ holds for $i \neq j$, we have:

$$\frac{p_ip_j - 1}{(p_i^2 - 1)(p_j^2 - 1)} < \frac{(p_i^2 + p_j^2)/2 - 1}{(p_i^2 - 1)(p_j^2 - 1)} = \frac{1}{2}\frac{1}{p_i^2 - 1} + \frac{1}{2}\frac{1}{p_j^2 - 1} \quad ; \text{ for } i \neq j \; .$$

This proves $z - 1 > 0$. For $i = 2, 3$ we have $p_1 < p_i$ and hence:

$$\frac{p_1p_i - 1}{(p_1^2 - 1)(p_i^2 - 1)} > \frac{1}{p_i^2 - 1} \quad .$$

Together with (16) we obtain:

$$0 \; < \; z - 1 < \frac{1}{p_1^2 - 1} - \frac{p_2p_3 - 1}{(p_2^2 - 1)(p_3^2 - 1)} \; < \; \frac{1}{p_1^2 - 1} \quad .$$

Since $z - 1 > 0$ holds, the lemma now follows from:

$$\begin{aligned} z - 1 \; = \; \frac{\prod_{i=1}^{3}(n/p_i - 1) - \prod_{i=1}^{3}(p_i^2 - 1)}{\prod_{i=1}^{3}(p_i^2 - 1)} \; &\geq \; \frac{\gcd\left(\prod_{i=1}^{3}(n/p_i - 1), \prod_{i=1}^{3}(p_i^2 - 1)\right)}{\prod_{i=1}^{3}(p_i^2 - 1)} \\ &\geq \; \frac{\prod_{i=1}^{3}\gcd(n/p_i - 1, p_i^2 - 1)}{\prod_{i=1}^{3}(p_i^2 - 1)} \quad . \end{aligned}$$

□

Lemma 25 *Let $T(d)$ be the number of integer solutions $0 \leq x < d$ satisfying $x^2 = 1 \pmod{d}$. Then we have*

$$\sum_{i=1}^{d} T(i) \; \leq \; 64 \cdot (d/24)^{5/4} \; .$$

Proof

We have $T(m \cdot n) = T(m) \cdot T(n)$ if $\gcd(m,n) = 1$. So we may represent the above sum as an Euler product. It is easy to show that for $s > 1$ we have:

$$\begin{aligned}
\sum_{d=1}^{\infty} T(d) \cdot d^{-s} &= \left(1 + 1 \cdot 2^{-s} + 2 \cdot 2^{-2s} + 4 \cdot 2^{-3s} + 4 \cdot 2^{-4s} + \ldots + 4 \cdot 2^{-\nu s} + \ldots\right) \\
&\quad \cdot \prod_{p \text{ prime}, p>2} \left(1 + 2 \cdot p^{-s} + 2 \cdot p^{-2s} + 2 \cdot p^{-3s} + \ldots + 2 \cdot p^{-\nu s} + \ldots\right) \\
&= \frac{1 + 1 \cdot 2^{-s} + 2 \cdot 2^{-2s} + 4 \cdot 2^{-3s}/(1-2^{-s})}{1 + 2 \cdot 2^{-s}/(1-2^{-s})} \cdot \prod_{p \text{ prime}} \frac{1 - p^{-2s}}{(1-p^{-s})^2} \\
&= \left(1 - 2^{-s} + 2 \cdot 2^{-2s}\right) \cdot \frac{\zeta(s)^2}{\zeta(2s)} \quad . \qquad (17)
\end{aligned}$$

To check the factor of the Euler product referring to the prime 2, note that $x^2 = 1$ has one solution in $\mathbb{Z}_2$, two solutions in $\mathbb{Z}_4$, and four solutions in $\mathbb{Z}_{2^\nu}, \nu \geq 3$. The above sum can effectively be computed by using Riemann's ζ-function.

Now, we put $d_0 = 19 \cdot 10^6$, $s = 6/5$. Then we obtain for $d > d_0$:

$$\sum_{i=d_0+1}^{d} T(i) \;=\; \sum_{i=d_0+1}^{d} \left(T(i) \cdot i^{-s}\right) i^s \;\leq\; \left(\sum_{i=d_0+1}^{\infty} T(i) \cdot i^{-s}\right) d^s \quad .$$

A numerical evaluation of the last sum, using (17), yields:

$$\sum_{i=1}^{d} T(i) \;=\; \sum_{i=1}^{d_0} T(i) + \sum_{i=d_0+1}^{d} T(i) \;\leq\; 204535220 + 2.54 \cdot d^{\,6/5} \quad ;\; \text{for } d > 19 \cdot 10^6 \; .$$

This proves the lemma for $d \geq 25 \cdot 10^6$. Smaller values of d are checked computationally. Note that $T(d) \cdot d^{-5/4}$ achieves its maximum at $d = 24$.

□

Lemma 26 *For $c \geq 2$ we have:*

$$\sum_{u=0}^{\infty} (u+c)^{-3} \;\leq\; 0.82\, c^{-2} \quad .$$

Proof

Since the second derivative of $(u+c)^{-3}$ with respect to u is positive for $u > 0, c > 0$, we obtain for $c \geq 2$:

$$\sum_{u=0}^{\infty} (u+c)^{-3} \;\leq\; c^{-3} + \int_{t=1/2}^{\infty} (t+c)^{-3} dt \;=\; c^{-2} \left(c^{-1} + \frac{c^2}{2(c+1/2)^2}\right) \;\leq\; 0.82\, c^{-2} \quad .$$

Note that $c^{-1} + c^2/(2(c+1/2)^2)$ is strictly monotonic decreasing for $c \geq 1$.

□

10 Summary

We have presented two probabilistic primality testing Algorithms SQFT and SQFT3. While Algorithm SQFT3 can be compared with the algorithms EQFTac and EQFTwc presented in [4], (as far as run time and error probability is concerned), Algorithm SQFT is much simpler than Algorithm SQFT3, but also asymptotically not as good as Algorithm SQFT3. For prime numbers and error probabilities as required in cryptography, we believe that Algorithm SQFT is sufficient in most cases of practical interest.

We recommend the use of Algorithm SQFT in devices with limited computational power for cryptographic applications such as prime number generation. It turns out that the requirements stated in [7] can be met with Algorithm SQFT and two test rounds in most cases of practical interest.

If computational power (especially program and storage size) is not critical, we can use Algorithms SQFT3. There are some special cases of prime candidates n (already mentioned in [4] for Algorithm EQFTac) where the algorithm is considerably more complicated than Algorithm SQFT.

Asymptotically, this extra expense can be neglected, since we have found that both algorithms have run time equivalent to $2t+1$ Miller-Rabin tests, when performing t test rounds. Some hints how to cope with this extra expense for test candidates of moderate size have been presented in section 5.

We also have stated worst-case and average-case bounds for the error probability of both algorithms in sections 6 and 7; and we have also analysed an incremental search version of Algorithms SQFT and SQFT3 in section 8.

References

[1] Jørgen Brandt, Ivan Damgård, On generation of probable primes by incremental search. In *Advances in cryptology CRYPTO '92 (Santa Barbara, CA, 1992), Vol. 740 of Lecture Notes in Comput. Sci.,* 358-370, Springer, Berlin, 1993.

[2] Jørgen Brandt, Ivan Damgård, Peter Landrock, Speeding up prime number generation. In *Advances in cryptology, ASIACRYPT '91 (Fujiyoshida, 1991), Vol. 739 of Lecture Notes in Comput. Sci.,* 440-449, Springer, Berlin, 1993.

[3] Ronald J. Burthe, Jr, Further investigations with the strong probable prime test. *Math. Comp. 65(213),* (1996), pp. 373-381.

[4] Ivan B. Damgård, Gudmund S. Frandsen, An Extended Quadratic Frobenius Primality Test with Average and Worst Case Error Estimates. *Series RS-03-8, BRICS, Department of Computer Science, University of Aarhus,* February 2003.

[5] Ivan B. Damgård, Peter Landrock, and Carl Pomerance, Average case error estimates for the strong probable prime test. *Math. Comp. 61(203),* (1993), pp. 177-194.

[6] Jon Grantham, A probable prime test with high confidence. *J. Number Theory 72(1),* (1998), pp. 32-47.

[7] ISO/IEC 18032: Information technology – Security techniques – Prime number generation, 2005.

[8] Arjen K. Lenstra, Eric R. Verheul, Selecting Cryptographic Key Sizes. In *J. Cryptology 14(4),* (2001), pp. 255-293.

[9] Louis Monier, Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theoret. Comp. Sci. 12,* (1980), pp. 97-108.

[10] Siguna Müller, A probable prime test with very high confidence for n = 1 mod 4. In *Advances in cryptology, ASIACRYPT 2001 (Gold Coast), Vol. 2248 of Lecture Notes in Comput. Sci.,* pp. 87-106, Springer, Berlin, 2001.

[11] Siguna Müller, A probable prime test with very high confidence for n = 3 mod 4. *J. Cryptology 16 (2),* (2003), pp. 117-139, Springer, Berlin, 2003.

[12] Carl Pomerance, On the distribution of pseudoprimes. *Math. Comp. 37,* (1981), pp. 587-593.

[13] Michael O. Rabin, Probabilistic algorithms for testing primality. *J. Number Theory 12,* (1980), pp. 128-138.